TWO ROADS TO NEXT-GEN COMPUTI

NG

In the next ten years, computing advances will face severe limitations. We can either try to overcome them or drive headlong into them.

MOST OF THE ELECTRICAL POWER GENERATED IN Los Alamos County goes to the national laboratory, with the largest chunk, about 25 percent of the county's power, going to the Lab's proton accelerator. But its next-largest power-hungry behemoth after that, consuming about 20 percent of the county's total production—roughly equal to the power consumed by all of Los Alamos's businesses and residences combined—is a collection of computers. These are supercomputers that, by virtue of their sophistication, are pushing up against serious power and performance limitations.

The Lab's high-performance computing (HPC) facilities primarily comprise more than ten supercomputers, one quantum-annealing computer, and several other HPC systems frequently used as test beds for experimentation. Lab scientists use these machines to perform computationally intensive simulations of many complex phenomena, from submicroscopic material changes inside the high-radiation environment of a nuclear weapon to the sensitive atmospheric dynamics that govern the global climate; from the evolution of cancer to the evolution of the universe. For a given top-of-the-line supercomputer, the performance of such simulations is largely limited by two things, one of which is the allotted electrical power.

The other limitation comes from an accumulation of tiny hardware glitches, called faults, often caused by natural radiation. These faults can randomly alter computer information, causing a zero to become a one, for example. Some faults are harmless, changing values that an application isn't using anyway. Others produce errors, manifesting as incorrect outputs or procedures. And occasionally one causes a crash, changing something so foundational that the operating system can no longer function without a full restart.

In personal computers, even those rare faults that get amplified to become crashes generally constitute little more than a nuisance nowadays. In supercomputers, however, a multitude of faults can develop in succession due to the sheer number of chips that can be affected. These constitute a serious drain on performance. Los Alamos is on the forefront of advanced supercomputing, pushing toward the exascale (10^{18} "flops," or floating-point operations per second, for those in the know). That's more than a hundred times faster than the world's current leading supercomputer and many millions of times faster than a new top-of-the-line laptop—and vastly more vulnerable to both faults and power insufficiency than either one.

Los Alamos scientists are pursuing two novel and somewhat contradictory paths to get around these limitations. One of these, as might be expected, involves tighter control over every bit of information: aggressively ferreting out faults and correcting them as they occur. The other, perhaps paradoxically, involves looser control of information: tolerating some randomness or reduced precision to save power and capacity for higher-precision operations elsewhere. Both strategies are likely to contribute to overall higher performance within power constraints and may ultimately find their way to consumer-market devices like phones and laptops, improving battery life and maybe, just maybe, eliminating those interminable hourglass and pinwheel icons from all of our lives.

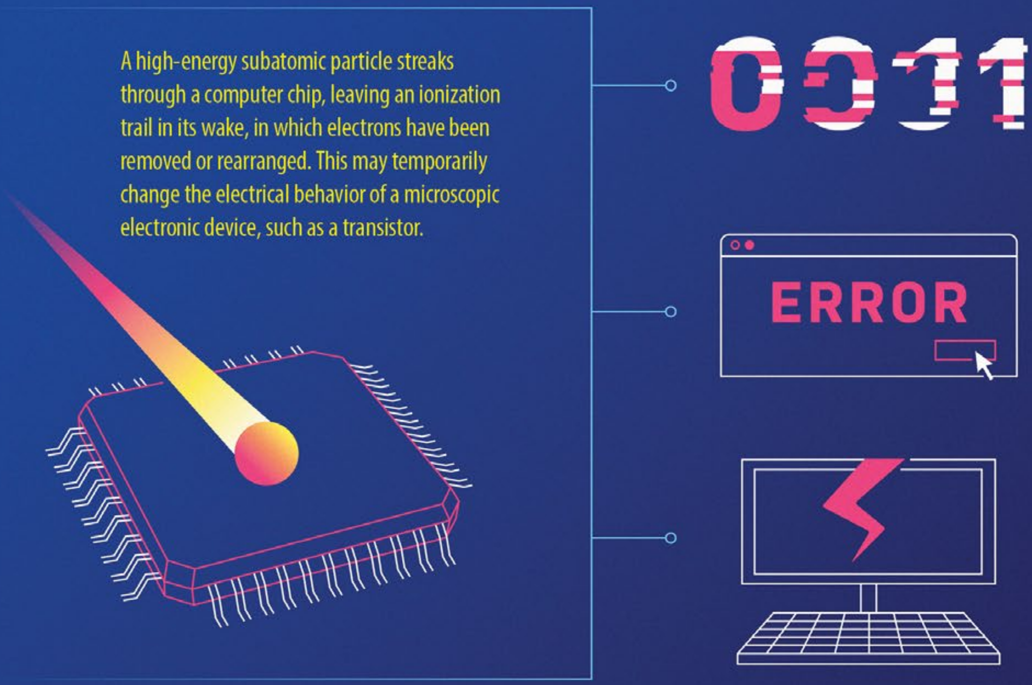
PACKING IN MORE TRANSISTORS WILL INCREASE A CHIP'S VULNERABILITY ALONG WITH ITS PERFORMANCE

No more Moore

Moore's Law famously asserts that the number of transistors per unit area that can be manufactured on a computer chip doubles every year or two—every year when the law was first postulated in the 1960s and every 18–24 months now. Historically, this has meant a steady increase in processing power, memory, and other measures of computer performance and has contributed strongly to economic growth around the world.

There have been many incorrect predictions of the end of Moore's Law, when the computing industry would no longer be able to count on its steady gains. At least one such prediction, however, seems very difficult to avoid, because with a higher transistor density comes a higher sensitivity to radiation-induced faults. A single stray neutron, streaking down from the upper atmosphere in the wake of a cosmic-ray collision with an air molecule, could pass through a computer chip and introduce a fault into not just one bit of information—something called a single-event upset, or SEU—but perhaps as many as 16 bits within a closely packed area of memory. When 16 zeros and ones change all at once, it's more likely that a calculation will be noticeably affected.

How to Flip a Bit



A high-energy subatomic particle streaks through a computer chip, leaving an ionization trail in its wake, in which electrons have been removed or rearranged. This may temporarily change the electrical behavior of a microscopic electronic device, such as a transistor.

FAULT
Usually a single-event upset (affecting information stored in memory) or a single-event transient (affecting some other circuit) involves a radiation-induced electrostatic discharge that causes a change in stored information or circuit activity.

ERROR
An incorrect value, calculation, or operation, possibly resulting in an error message: one or more applications generate incorrect values or do something they're not supposed to do.

CRASH
An unrecoverable error that renders the computer unusable until it has been powered down and restarted.

Such multiple faults also make it more difficult to successfully implement a remedy known as an error-correcting code (ECC). Typically, that code works by appending additional, redundant bits to a number, so that a value of zero, say, might become a zero followed by two additional zeros. If, somewhere down the line, the value is no longer 0 00, but perhaps 0 10 or 1 00, then the ECC knows something is wrong. It can fix the problem by making the most likely assumption, that of a single bit flip—in this case, the 1 should be a 0.

But if 16 bits change at once, this becomes much more difficult to do. If every bit of information requires two redundant bits just to safeguard against the simplest kind of fault, an SEU, then computational resources would be much more seriously taxed by having to provide enough redundancy to fix severe multiple-event upsets. Short of building in an enormous amount of redundancy and taking an enormous hit in terms of power and performance, there would be no way of knowing what had been corrupted. Some of the corrupted bits might even reside in the error-correcting operation itself.

Heather Quinn and her colleagues in the Los Alamos Space Data Science and Systems group have spent years working to understand and combat radiation effects on computer chips. Until recently, it has been very specialized work because standard ECCs could do a satisfactory job for computers on the ground; but on aircraft and spacecraft—that is, closer to the cosmic rays—and in other high-radiation environments, computer systems would be particularly vulnerable to serious, uncorrectable errors. Indeed, the Cibola Flight Experience satellite, which has been studying Earth's ionosphere for more than ten years, successfully uses specialized Los Alamos hardware and software to recover from SEUs that occur on a daily basis.

"We've made a lot of progress locking down the SEU issue," says Quinn. "But Moore's Law is taking us to a place where SEU correction will not be enough. Packing in more transistors will increase a chip's vulnerability alongside its performance." She points out that while the danger is more pronounced in space-based systems, it will soon be a major problem on the ground for consumer electronics—ten years by most projections and even sooner for HPC. "But if we can come up with a solution for space," she says, "where radiation is abundant and electrical power is at a premium, then we should be able to use that for HPC and other computing environments on the ground."

Trying Trikaya

Specialized computing systems have some form of fault protection now. HPC systems have a similar safeguard to what most people probably use in their own personal computing: they save often. That is, at various checkpoints, they record a last-known state to be restored in the event of a serious fault event. Along with servers and high-end desktop computers, HPC systems also have a form of ECC protection known as a SECDED code ("single event correct, double event detect"). The redundant bits are checked as values are moved in and out of memory—at a significant cost to power, memory, and chip space. (Laptops, tablets, and phones generally forego these protections in favor of speed and battery life.) But as Moore's Law approaches its limit and faults begin to arrive en masse, SECDED codes (as well as more cumbersome double-event-correcting codes) will become less effective. That's why Quinn and collaborators have been working on a broader approach.

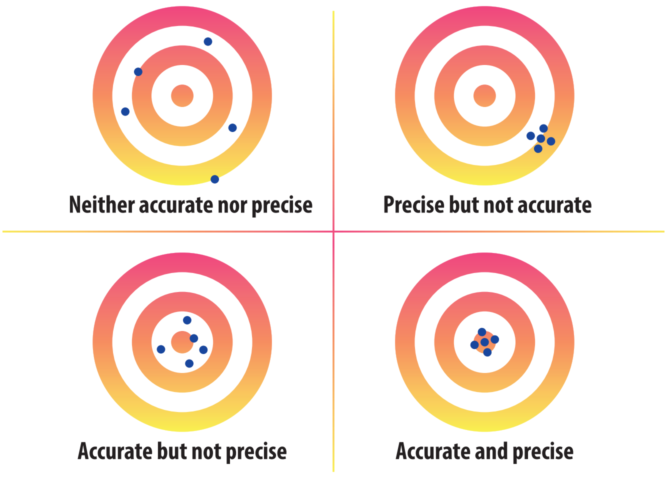
When a computer program is written, or coded, it is written in a programming language that expresses logical instructions: arithmetic operations, calls to subroutines, if-thens, do-whiles, and the like. The program is then "compiled" by another program (called a compiler) to convert the human-comprehensible code into machine instructions. Quinn's approach is essentially to force the compiler to build programs such that they operate in triplicate, using a technique that she validated for a sweeping new ECC called Trikaya. (The name refers to a Buddhist doctrine recognizing Buddha's three "kayas," or bodies.)

A simpler version of this approach might cover a computation, such as adding two numbers. If this is done three times, and the results are 4, 4, and 79, then an ECC could assume that a fault occurred in the third calculation because it lost a "best two out of three" contest. Essentially it was outvoted: two votes for "4" and only one vote for "79" strongly suggests that a fault disrupted the final calculation. (Alternatively, an identical fault event could have affected the first two calculations in exactly the same way, but this is an extremely unlikely scenario.) Trikaya extends this three-vote concept to full, simple programs, with more advanced capabilities planned. In effect, everything the software does, it does three times, and then it takes a vote to establish the results. Or, to save power and other computational resources (e.g., access to hardware also needed for other tasks), it might do everything twice and only conduct a third vote if the first two disagree.

SOME RESOURCE-INTENSIVE OPERATIONS DO NOT REQUIRE PERFECT DETERMINISTIC COMPUTATIONS

Theoretical work at Los Alamos over the past several years has verified Trikaya's algorithms, and operational testing is currently underway at the Los Alamos Neutron Science Center—the same proton accelerator and neutron source that typically consumes a quarter of Los Alamos County's electrical power. It can spray neutrons on a chip and cause faults at a much higher rate than would happen naturally, even in space. That way, researchers can test how reliably Trikaya actually fixes errors caused by radiation. If successful, Trikaya protection will likely appear on the next generation of satellites tasked with detecting nuclear detonations from space. And if it proves successful there, it may influence future microprocessor designs and ultimately filter down to the personal-computer market.

Still, Trikaya, like earlier fault-protection strategies, is resource intensive. It depends on repeating computations and therefore saps power and performance. It may protect satellites, and maybe personal computers, but by itself,



Unlike in conversational use, accuracy and precision mean different things in technical use. Accuracy refers to the ability to produce a correct or optimal result. Precision refers to limiting the amount of variation in a result. (Precision can also refer to the number of significant digits or decimal places in a calculated or measured value, since a measurement of 3.28 could mean 3.277 or 3.28412, and these represent variation in the result.) In inexact computing, fault vulnerability and power consumption may be reduced by deliberately restricting either the accuracy of a computation (by using an approximation) or its precision (by restricting the number of significant digits or using probabilistic methods).

it doesn't solve the HPC resource problem. For that, Quinn collaborates with Los Alamos colleague and HPC expert Laura Monroe.

Inexact computing

"Not every problem requires an exact answer," says Monroe. "Sometimes it's good enough—or even better—to use inexact computing methods to get a high-probability, close-enough answer faster and with less power than an exact answer."

Wait, what? Isn't the whole point of computers that they are reliable and calculate correct answers lightning fast?

Well, yes, that *was* the point. But as the Moore's Law crisis approaches, and with HPC systems already resource-strained, the situation has changed. In addition, certain resource-intensive operations, such as image processing, social networking, searching, and sorting, genuinely do not require perfect, deterministic computations. And other emerging areas of advanced computing that require fast-but-not-perfect calculations, such as machine learning and perception, are likely to operate as well or better with inexact algorithms. Even some traditional computations, including, ironically, error correction, appear to be amenable to inexact methods. It is now an important research challenge to understand which problems might benefit from the speed and cost savings of an inexact approach.

"Research into inexact computing started in the early days of computing as an approach to unreliable systems," says Monroe. "But that need abated with the move away from vacuum tubes to more reliable transistors and integrated circuits in the 1950s. Now it's becoming important again."

SOMETIMES GUESS-AND-CHECK IS FASTER THAN DOING THINGS THE USUAL WAY

Inexact computing consists of two similar-sounding but distinct approaches: probabilistic computing and approximate computing. Probabilistic computing, like a large number of coin flips to converge on 50-50 odds, is non-deterministic, meaning that the computational paths are different each time a computation is performed, and the results may well be different too. Approximate computing is deterministic and straightforward, but it is abbreviated, such as by reducing the numerical precision (e.g., number of decimal places) or shaving off iterations in a looping calculation.

Another way to characterize these approaches involves the difference between accuracy and precision. Accuracy is the correctness of a calculation (or measurement), or how close it is to the actual answer. Precision is how close each calculation (or measurement) is to other calculations of the same problem. So, accuracy or precision—for inexact computing purposes, it is possible to relax either one.

For non-resource-limited computers today, both accuracy and precision are widely understood to be at acceptable levels, with both limited only by the finite number of bits used in a digital-computer calculation. Most calculations are brute-force applications of cumbersome but straightforward arithmetic carried out with many more decimal places than are needed. And because of the deterministic nature of the hardware and software used, if a calculation is repeated, exactly the same answer will emerge.

As a conceptual example, consider the irrational number pi, which requires an infinite number of decimal places to express it exactly. To calculate it, a computer could carry out a formula to add up an infinite but converging series of decreasing terms. (See "Two Slices of Pi" on the next page.) After adding up a huge number of terms, pi is guaranteed to be correct up to some number of decimal places. One could then choose to trade accuracy for reduced resource consumption (power, time, hardware requirements) by choosing to sum only the first few terms. The precision would be perfect (to however many terms one chooses to add), in the sense that repeated calculations would always yield precisely the same answer, but the answer would be off from the true value of pi, so the accuracy would be relaxed. This is an example of approximate computing.

Alternatively, one could choose an accurate method with precision relaxed—namely, the exact geometric definition of pi, the ratio of a circle's circumference to its diameter—but use an imprecise, probabilistic computation to get there. The computer could generate random numbers to represent pixels and basically draw a circle from which to obtain pi; the more pixels, the more closely the estimate will approach pi. One could design this process with enough pixels that the result is *x* percent likely to be within *y* percent of the correct value. The answer would be different each time the calculation is performed, but a large number of trials would converge toward the exact answer.

This is an example of probabilistic computing (not one used in practice to calculate pi). For some tasks, including many HPC simulations, such probabilistic and approximate algorithms may be sufficiently accurate and precise, and yet considerably less resource-intensive than a standard deterministic computation.

Turn up the radio

Some probabilistic computing requires specialized hardware. Prototypes from academia, industry, and government already exist, delivering impressive improvements in speed and power use. Image processing, in particular, has proven to be fertile ground for the potential benefits of probabilistic computing.

Why image processing? "There are really two reasons," explains Monroe. "Most obviously, images don't always need to be perfect; sometimes minor variations will make no difference in how the image is used, even in scientific contexts. In addition, existing image-processing algorithms can be very resource-intensive, so there's a lot of opportunity for savings."

Two Slices of Pi

Which is the better way to compute the value of pi—an irrational number that can only be expressed exactly with an infinite number of decimal places—a deterministic or probabilistic calculation?

Each version below will be based on a ¼ slice of pi, or $\pi/4$.

DETERMINISTIC APPROACH

Because there's no direct way to calculate pi as it is defined—the circumference of a circle divided by its diameter—computers must calculate pi by some other method. As an instructive example, consider that the tangent of $\pi/4$ radians (equivalent to 45°) is 1: $\tan(\pi/4) = 1$.

By taking the arctangent (inverse of the tangent) and multiplying by 4 on both sides of that equation, one gets: $\pi = 4 \times \arctan(1)$.

The arctangent (and many other functions) can be expressed as a power series—an exact formula based on the sum of an infinite number of terms. For the arctangent, this is:

$$\arctan(x) = \frac{x^1}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \ldots$$

so the formula for π, with x set to 1, becomes

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots \right)$$

After just one term, we have $\pi = 4$. After two terms, it's $\pi = 4 \times (1 - 1/3) = 2.666666667$. After 3 terms, it's $\pi = 4 \times (1 - 1/3 + 1/5) = 3.466666667$, and so on. Because the terms get smaller and alternate signs (plus, minus, plus, minus), each additional term helps to zero in on the correct value (3.14159…). A computer program could meet reduced power and computational resource requirements by adding only the first few terms (the first ten? hundred? thousand?) and ignoring the infinite number of terms that follow.

The result would have good precision, in the sense that repeated calculations adding the same number of terms give the exact same result, but it might not be accurate enough—that is, not close enough to the correct value of pi for a given task. In fact, the arctangent power series nicely illustrates this limitation because it converges very slowly; an enormous number of terms would have to be added to arrive at a reasonably accurate approximation of pi. Fortunately, other power series, with terms that also alternate signs but diminish much more rapidly, can be used for calculating pi.

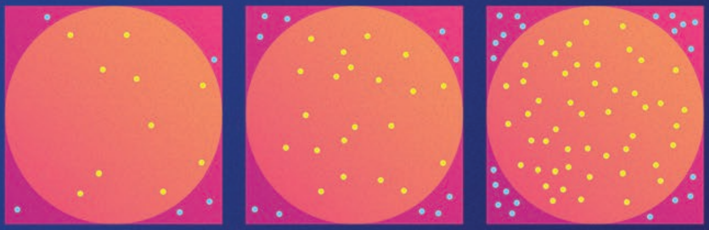
PROBABILISTIC APPROACH

Alternatively, one could develop an exact geometric method and use random numbers to make the irrational-number computation tractable. For example:

The area of a circle is πr^2.

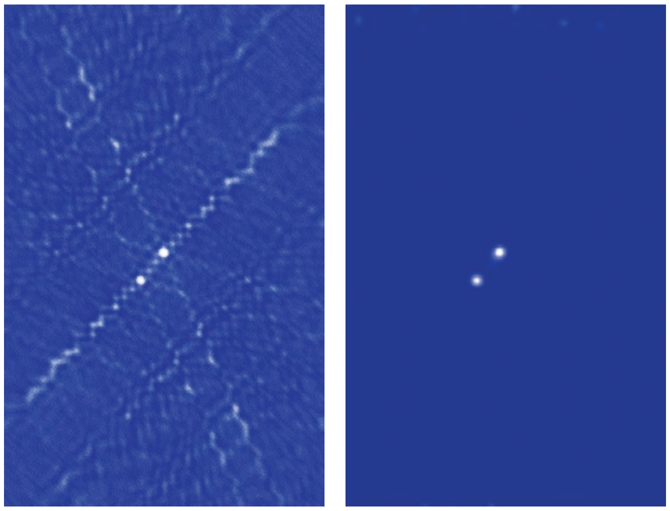
The area of a square just barely containing that circle is the length of its side, squared; here the side length is just the circle's diameter, or twice its radius, $2r$. Therefore the ratio of the area of the circle to that of the square is:

Area ratio (circle's area/square's area) = $\pi r^2 / (2r)^2 = \pi r^2 / 4r^2 = \pi/4$.
Equivalently, $\pi = 4 \times$ (circle's area/square's area).



A computer capable of generating random numbers could place dots at random pixel locations within the square. After some number of pixels is placed, it could divide the number inside the circle by the total number inside the square (including those inside the circle). Pi should then equal four times that amount. Just as flipping a coin many times causes a 50–50 probability to emerge, more dots should lead to a more accurate result.

That result would have limited precision because repeated calculations, with different random numbers each time, would yield different results. But with enough dots, such variations would be small, and there would be a high degree of probability of a highly accurate result (close to 3.14159…). In some applications, calculating with random numbers like this can be better (less power, more speed, better fault tolerance, etc.) than an equivalent deterministic calculation.



Certain resource-intensive image-processing tasks, such as "cleaning" a radio-astronomy image as seen here, can be sped up with an inexact-computing algorithm. In this case, a probabilistic approach to a bottleneck in the calculation yields a noise-reduced image more quickly and efficiently than a deterministic approach.

CREDIT: Bill Junor/LANL

For example, when doing image processing for radio astronomy, one wants to keep the extraterrestrial radio sources and remove the radio noise. This is typically done with an iterative algorithm called CLEAN. You start by identifying the *x* brightest pixels. Then you apply a mathematical operation called a Fast Fourier Transform (FFT) to develop a "point spread function" for those pixels. That's just what it sounds like: a way to take a perfect point source (of radio waves in this case, such as a quasar) and calculate how that point will spread into neighboring pixels in the instrument recording the image. Then the resulting blurred bright spots are subtracted from the original image and the whole process is repeated many times. When done, all that remains in the image should be noise. Then, finally, the saved bright spots are combined with a noise-subtracted background that hopefully, but not for certain, displays a good image of real astronomical radio sources. Whew.

Radio telescopes are usually located in remote locations far from sources of interference, where power is at a premium. Good computational performance is therefore essential, and a graphics processing unit (GPU) is a reasonable choice because it is designed for parallel processing—subdividing computations into pieces that can be carried out simultaneously—provided that parallel algorithms exist to take advantage of this capability. At the time Monroe started to look into inexact methods for image processing, there was a good parallel algorithm for FFT on a GPU, but selecting the *x* brightest pixels was a bottleneck.

Monroe and collaborators (including Joanne Wendelberger, whose work is also featured on page 5, and Sarah Michalak) developed an alternative, probabilistic approach for the pixel-selection problem on a GPU. They used a randomized process to select "pivots": values for pixel brightness with a user-chosen probability of bounding the *x* brightest pixels and were used to sort pixels into bins. By repeating this randomized sorting procedure to do the bright-pixel selection, they were able to improve the corresponding processing times by a factor of 1.5–6 and remove the bottleneck for the CLEAN algorithm. This also increased capacity to handle images four times larger than previous methods.

"It's basically a Las Vegas algorithm—even though it is probabilistic, it always gives the correct answer," says Monroe. "Sometimes, for the right problems, guess-and-check can be faster in practice than doing things deterministically."

So far, all these advances—probabilistic chips, probabilistic algorithms, and advanced error correction like Trikaya—are being created for special-purpose computations, not for the general computing market. General consumer computing will likely follow as new chips become increasingly susceptible to radiation-induced faults. But one inexact-computing improvement may be relatively easy to come by right now.

Many programmers today carry out deterministic math with higher decimal-place precision than necessary. "Floating point" decimal numbers used in software coding carry either single precision, which is encoded in 32 bits, or double precision, which is encoded in 64 bits. Yet so many bits (and corresponding decimal places) may not always be needed. For example, Monroe is working with Lab colleagues to develop codes that can be improved in terms of the tradeoff between quality and resource use simply by going from double to single or even half precision in some parts of the code. For HPC, that would mean saving precious power or doing more without consuming any additional power. It would also mean saving memory and storage space. And it would mean faster performance, or at least offsetting the performance reduction caused by the advanced ECCs that will soon be necessary.

It's a little thing, limiting numerical precision, but every little bit helps when you're under constant assault by radiation from space. LDRD

—Craig Tyler

More **high-performance computing** at Los Alamos

- **Facilities and capabilities**
  http://www.lanl.gov/org/padste/adtsc/hpc/index.php
  http://www.lanl.gov/projects/advanced-simulation-computing/index.php
- **Quantum computing**
  http://www.lanl.gov/discover/publications/1663/2016-july/not-magic-quantum.php
- **Exascale computing for materials science**
  http://www.lanl.gov/discover/publications/1663/issues-archive/june2012.pdf
- **History of supercomputers at Los Alamos**
  http://www.lanl.gov/discover/publications/alumni/2015-01/science-100supercomputers.php
- **Select supercomputer applications**
  Climate science: http://www.lanl.gov/discover/publications/1663/2016-december/ice-sheet-demise.php
  Cancer genetics: http://www.lanl.gov/discover/publications/1663/2016-december/what-causes-cancer.php
  Petascale cosmology: http://www.lanl.gov/discover/publications/1663/issues-archive/october2012.pdf
  Ocean ecology: http://www.lanl.gov/discover/publications/1663/2013-july/the-bottom-of-the-ocean-food-chain.php

More@LANL